Edging Towards Exascale With NIX

Jim McKie
Bell Laboratories
jmk@research.bell-labs.com

John Floren
Sandia National Laboratories
jfflore@sandia.gov

Abstract

This paper describes NIX, an operating system for manycore CPUs. NIX has been influenced by our work on Blue Gene, Roadrunner, and more traditional HPC systems such as clusters. NIX is designed to support a heterogeneous model in which some CPUs can run a kernel and some can not; in which not all memory is shared; and in which the architectures might not all be binary compatible.

Introduction

The anticipated architecture for future systems is a heterogeneous multicore model, where some cores are incapable of running a full operating system in order to meet power/density goals and memory constraints. We have developed a hybrid runtime prototype called NIX to explore alternative design points for such architectures. NIX partitions cores by function, amongst which are Timesharing Cores, or TCs, and Application Cores, or ACs. One or more TCs run traditional applications, functioning in a way indistinguishable from a standard timesharing kernel. ACs are devoted to running an application with no interrupts; not even clock interrupts. NIX is capable of shipping unsupported events on ACs, such as exceptions and requests for OS/R services, to a TC running on a core capable of handling such events. The number of cores assigned to each function need not be not static and can change as needed, the only requirement being that there must always be at least one TC. An important design feature of NIX is the incorporation (or retention) of a conventional timesharing core, thereby providing backwards compatibility, and a way to quantify and evaluate design points.

This has been used to explore the design space for designating *roles* to cores for particular duties, experimenting with different mechanisms for handling exceptions and requests for OS/R services, and evaluating different mechanisms for efficient core to core communication. Initial results using the FTQ OS Noise benchmark show the NIX runtime dedicated ACs achieve a much better noise profile than a traditional kernel, in line with that of the best FTQ results obtained by current practice, e.g. a light-weight kernel (LWK) such as the BlueGene CNK.

Some of the issues to be faced in exploiting future systems using the aforementioned methodology are shown by a naïve implementation of an HTTP server and a networked key-value store written to examine performance. In both applications, a process is assigned to each new client connection and handles both I/O and request processing, and performance was acceptable and similar when running on two traditional timesharing systems, Linux and NIX using only TCs. However, there is significant performance loss when run on ACs due to the additional overhead of sending I/O to another core. This simple analog of attempting to use heterogeneity shows some of the challenges ahead for both application writers and hardware designers: for example, applications could implement strategies to reduce the number of I/O operations, and I/O hardware could be self-virtualized, but both approaches have costs and issues in scalability, e.g. will the hardware approach scale if there are 1000s of cores per node? Early identification of potential problems is important.

We have also investigated how this architecture intersects and affects traditional OS/R components. Amongst these are the process execution model, interprocess communication and synchronization, page size and management, and memory allocation (including NUMA).

Status

NIX is currently a modified timesharing system that has previously been successfully run on systems ranging from embedded to HPC. The application of the ideas in NIX are not, however, tied to any particular OS and have been taken up by other groups working in the same area (e.g. FusedOS[1]). In addition, experience with NIX is being used to guide the design of Osprey[2], a clean-slate runtime for predictable cloud computing being developed by the Network Systems group at Bell Labs.

NIX was created as a joint effort by Sandia National Laboratories, Bell Laboratories, and The Universidad Rey Juan Carlos, Madrid. All three contributed to the initial design and implementation, and continue to be actively involved. Sandia is currently focused on benchmarking and applications, Bell Labs on basic OS/R principles, hardware support, and execution models, and UJRC have recently targeted scheduling issues and contributed much of the instrumentation and monitoring scaffolding which is central to understanding the combined effects of OS/R and exotic hardware design changes.

The initial NIX hardware target was a manycore AMD64 system. It is anticipated that the initial core port of NIX to BG/Q, including full memory-management support and user-environment, will be completed in Summer 2012, with full multi-core and multi-thread support later in the year. Once the core port is in place, the team will move on to integrated networking and multi-node support.

Discussion

While NIX is a new operating system, we have chosen to build it on a traditional operating system in a way that preserves binary compatibility but also enables a sharp break with the past practice of treating many-core systems as traditional SMP systems. We were influenced in this decision by watching the struggles of a number of "from scratch" operating systems efforts; we wanted to always have a working system on which to build.

The challenges of exascale force a rethink on many components of the OS/R, but there are many tricky implementation decisions that have been made correctly in extant kernels, and many of the components and techniques *will* transfer forward; there is no need to reimplement and risk recreating the same errors that were resolved in an existing OS/R until the design space is better understood.

NIX is also strongly influenced by our experiences over the past five years modifying and optimizing a general-purpose OS to run on HPC systems such as IBM's BlueGene, applying our experience with large scale systems to general purpose computing. This experience[3] has shown that *Right-Weight Kernels* in HPC can have negligible overhead compared to specialized runtimes, while offering all the advantages of access to a general-purpose OS; in this light, the design decision to use a TC seems sensible, and allows us to concentrate on the interesting and novel architecture challenges.

Challenges and Future Work

Given its heritage and structure, NIX obviously targets many legacy OS/R issues. However, the emphasis on having a well-understood and instrumented system enables NIX to evaluate comparative OS/R capabilities rather than pure performance in other key challenge areas, for example:

- Hardware. Many opportunities will exist to leverage new hardware facilities such as transactional memory support, atomic memory regions, etc. to enhance performance, efficiency, and reliability of the OS/R in support of target applications;
- Managing heterogeneity at many levels. It is expected this will cause ongoing refinement of the core role mechanism, with extensions of the implementation into different process contexts;
- Scheduling within and between cores and nodes. Experiments have already identified scheduling anomalies when there is a dynamic mix of cores with different roles; such knock-on effects are a barrier to effective parallelism and scalability;
- Global control and coordination. The current model of one machine, one application is unlikely to survive; the move to a workflow model of computation, and the need to support fault management, will require the addition of scalable management across the applications and system, in much the same way as a traditional OS/R manages its resources.

Acknowledgements

This material is based upon work supported in part by the Department of Energy under Award Number DE-FC02-10ER25997/DE-SC0005158.

NIX is a collaborative effort and we would like to acknowledge the contributions of all the authors in [4], especially Ron Minnich, amongst others.

References

[1] Robert W. Wisniewski, Todd Inglett, Yoonho Park, Bryan Rosenburg, Eric Van Hensbergen, Kyung Dong Ryu. FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment. *Submitted to Supercomputing 2012*.

[2] Jan Sacha, Jeff Napper, Hening Schild, Sape Mullender, Jim McKie. Osprey: Operating System for Predictable Clouds. *The Second International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology,* June 2012.

[3] Ronald G. Minnich, Matthew J. Sottile, Sung-Eun Choi, Erik Hendriks and Jim McKie. Right-Weight Kernels: An Off-the-Shelf Alternative to Custom Light-Weight Kernels. In *ACM SIGOPS Operating Systems Review*, 2006, vol. 40, no. 2, pp. 22-28.

[4] Francisco J. Ballesteros, Noah Evans, Charles Forsyth, Gorka Guardiola, Jim McKie, Ron Minnich, Enrique Soriano. NIX: A Case for a Manycore System for Cloud Computing. In *Bell Labs Technical Journal*, 2012, Vol. 17, No. 2.